# Lab 7: Stored Procedures

**Pair Programming**
You may choose to work with a partner on this lab. This lab is the most challenging thus far, so we are encouraging (but not requiring) pair programming.

**Objective**
The goal of this lab is to introduce you to stored procedures. You will be creating a few stored procedures that will allow users to create, update and delete order details in the Northwind database. These stored procedures will enforce some simple business logic.

**Resources**
The example stored procedures get_Order Details_1 and update_Order Details_1 found in the file **SprocExamples.txt** (in the Lab Instructions folder on Angel) provide examples of coding conventions, validating parameters, raising errors, and appropriate comments.

**Turn in Instructions**
Place your .sql files for creating your stored procedures in a .rar (or .zip). **Also include a text file named who.txt listing you and your partner's names.** Place your archive in the Lab 7 drop box.

**Assignment Details**

1) In your copy of Northwind, you will first create basic frameworks for the stored procedures. You can do this either by using the Create Procedure Basic Template in the Template Explorer(View> Template Explorer) within Server Management Studio ***or*** by using the following queries:

```
CREATE PROCEDURE [insert_Order Details_1]
      (@OrderID_1       [int],
       @ProductID_2     [int],
       @UnitPrice_3     [money] = NULL,
       @Quantity_4      [smallint],
       @Discount_5      [real] = 0)
AS
INSERT INTO [Northwind].[dbo].[Order Details]
       ( [OrderID], [ProductID], [UnitPrice], [Quantity], [Discount])
VALUES ( @OrderID_1, @ProductID_2, @UnitPrice_3, @Quantity_4, @Discount_5)
GO
```

```
CREATE PROCEDURE [update_Order Details_1]
      (@OrderID_1       [int],
       @ProductID_2     [int],
       @NewQuantity_4   [smallint]= NULL,
       @NewUnitPrice_3 [money] = NULL,
       @NewDiscount_5 [real] = NULL)
AS
-- Update Order Detail values
UPDATE [Northwind].[dbo].[Order Details]
SET [Quantity] = @NewQuantity_4, [UnitPrice] = @NewUnitPrice_3, [Discount]
= @NewDiscount_5
WHERE ( [OrderID] = @OrderID_1 AND [ProductID] = @ProductID_2)
GO
```

```
ALTER  PROCEDURE [delete_Order Details_1]
      (@OrderID_1       [int],
       @ProductID_2     [int])
AS
--Delete the row with the given OrderID and ProductID in the OrderDetails
table
DELETE [Northwind].[dbo].[Order Details] WHERE ( [OrderID]    = @OrderID_1
AND [ProductID]= @ProductID_2)
GO
```

2) The stored procedure [delete_Order Details_1] simply takes an order ID and product ID and deletes the corresponding record from the Order Details table. When an item is deleted from Order Details, the company wants to adjust the quantity in stock for the product to reflect that the units weren't sold after all. Your first task is to refine the stored procedure to account for this.

   You should modify the stored procedure. One way to do so, is to expand your copy of Northwind, then expand "Programmability" and "Stored Procedure". Right click on "delete_Order Details_1" and select "Modify".

   a. Your procedure must be well documented. It should have comments for

major subsections. It should also include a header block describing the purpose of the procedure, giving an example usage, your name(s), and the current date. See the stored procedures listed in Resources above for examples.

b. Your procedure should validate parameters. It should return an error code (a non-zero result) and print a message if the parameters are invalid. Parameters are valid if the given order ID appears in the table and if the given product ID appears in that order.

c. Your procedure should return 0 if the delete is successful, otherwise it should return an error code and print a message.

3) The stored procedure [update_Order Details_1] wizard takes all the data for a single row in the Order Details table, plus an additional order ID and product ID corresponding to the primary key. The stored procedure finds the row with the matching primary key and updates that row's values to match those passed to the procedure.

   Your task is to refine the stored procedure to meet the following additional requirements:

   a. The OrderID and ProductID of a row cannot be changed. That is, the procedure should take just one of each argument and should use those arguments to find the row to be changed. Only the Quantity, UnitPrice, and Discount of that row can be changed. (If an end user wanted to change the OrderID or ProductID, she would have to delete the row and add a new row.)

   b. The Quantity, UnitPrice and Discount parameters should be optional. One way to do this is to set default "magic" values for the parameters in the procedure declaration, like **@UnitPrice [money] = 0**. Then you can check whether the actual value of @UnitPrice matches the default, indicating that the argument was omitted. You should choose default values that won't actually be used. (Consider, can you really do that?)

   c. When updating the record, do not change Quantity, UnitPrice and Discount values unless new values were provided.

   d. Adjust the quantity in stock for the product by adding the old quantity back to inventory and subtracting the new quantity from inventory.

   e. However, if there is not enough of a product in stock, then abort the stored procedure without making *any* changes to the database.

   f. Print a message if the quantity in stock of a product drops below its Reorder Level as a result of the update.

   In addition to satisfying the functional requirements above, make sure that your stored procedure is well documented, validates parameters, and returns appropriate result codes as in the previous task.

4) The stored procedure [insert_Order Details_1] takes all the data for a single row in the Order Details table and adds a row with those values to the table.

Your task is to refine the stored procedure to meet the following additional requirements:

a. Make the UnitPrice and Discount parameters optional, using the technique discussed in the previous task.

b. If no UnitPrice is given, then use the UnitPrice value from the product table.

c. If no Discount is given, then use a discount of 0.

d. Adjust the quantity in stock for the product by subtracting the quantity sold from inventory.

e. However, if there is not enough of a product in stock, then abort the stored procedure without making *any* changes to the database.

f. Print a message if the quantity in stock of a product drops below its Reorder Level as a result of the update.

In addition to satisfying the functional requirements above, make sure that your stored procedure is well documented, validates parameters, and returns appropriate result codes as in the previous tasks.

5) Complete the anonymous feedback for Lab 7 on Angel.

**Turn in Instructions**

Place your .sql files for creating your stored procedures in a .rar (or .zip). **Also include a text file named who.txt listing you and your partner's names.** Place your archive in the Lab 7 drop box.

**Revision History**

Jan 14, 2006: Minor editing by Curt Clifton
Jan 10, 2006: Revised for SQL Server 2005 by Eliza Brock
Jan 11, 2006: Revised by Curt Clifton and Steve Chenoweth
Jan 10, 2006: Written by Will Mathies